新人技術者のための

最終回

ロジカルシンキング入門

第10回

「工学の知」とは何か

冴木 元

システムの記事

ビギナーズ

最終回は，本連載のタイトルとなっている「ロジカル・シンキング」という物の考え方について解説する．ロジカル・シンキングは，エンジニアに不可欠な物の考え方そのものである．

（編集部）

Gさんは，あるソフトウェア会社の課長です．彼は，現場一線のエンジニアだったころは「火消し役」として，あちこちの開発チームから重宝されていました．徹夜で何千ステップものプログラムを仕上げ，しかもバグがほとんどないという彼の技術力は，いろいろな人から頼りにされていたのです．

「現役」時代はその腕力にものを言わせてトントン拍子に出世したGさんでしたが，晴れて管理職となった今では，実際にプログラムを作成することはなくなりました．今ではもっぱら，足を棒にして営業に回る毎日です．

今日は，忙しい営業活動の合間をぬって，組み込みシステム関連の展示会に来ているところです．いろいろと展示を見てまわったGさんはその日の終わりに，ある講演会に参加しました．そこでは，名の知られたITコンサルタントが「ユビキタス社会における今後の戦略」について熱弁を振るっていました．

講演を聴き終わったGさんは不満げです．現場たたき上げのGさんは，この手の議論に日ごろから懐疑的だったからです．要するに，経営戦略論につきものの「ロジカル・シンキング」などといった問題解決のテクニックの類を耳にすると，Gさんのおへそは大いに茶を沸かすのです．

「あいつの話はどうもまゆつばものだなあ…．おれが足で集めた情報と照らし合わせると，いろいろなほころびが見える．プレゼンそのものはうまいが中身がない．やはりあの手のロジカル・シンキングとやらはくせ者だ．…おっと，今日の接待は何時からだっけ？」

今日は大切なお客さまを接待する日です．Gさんは足早に予約してある飲食店に向かうのでした．

● 経営管理手法の一つである「ロジカル・シンキング」

2006年6月号から続いたこの連載も今回で最終回を迎えますが，冒頭のGさんのような人には理解してもらえなさそうな記事を最初から最後まで書き続けてしまったような気がします．

上記に述べたように，ロジカル・シンキングというのは，もともとは経営戦略立案のためのテクニックの一つです．大きな書店の経営書のコーナに行くと，「ロジカル・シンキング」と題した，似たような内容の書物がいくつも並んでいます．ロジカル・シンキングという考え方は，経営書の世界では使い古された概念なのですが，知らないエンジニアの方も多いと思います．

最終回となる今回は，今までロジカル・シンキングとして紹介してきた物の考え方を，いわば総論的に紹介します．また，MECE（ミーシーと発音）と呼ばれるロジカル・シンキングのキー・コンセプトや，ロジカル・シンキングというアプローチ自体が持っている限界についても解説します．

● 「いきなり解決策に飛びつかない」がポイント

ロジカル・シンキングの出発点となる重要な考え方は，

KeyWord　ロジカル・シンキング，問題の切り分け，MECE，樹形図，fact-based analytical approach，テスト，ソフトウェア工学
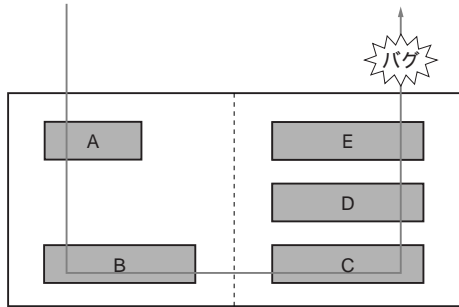
図1　流れで全体像をとらえ，切り分けを進める

例えば，「音が出ない」，「画像が出ない」，「入力の応答が返らない」などの事象から問題のブロックを見付け，根本的なバグを突き止めるには，このようなアプローチが必要．

「問題を見付けたとき，いきなり解決策に飛びつかない」ということです．「問題を見付けたら，まず根本原因を探ること」と言い換えてもよいでしょう．

開発中のシステムが突然動かなくなったら，エンジニアはどう行動するでしょうか．「どうしたら動くだろう」と考えて，とにかくいろいろ試してみるでしょうか．経験を積んだエンジニアであれば，「とにかくあれこれといじってみる」というアプローチがまずいことはお分かりかと思います．なぜなら，動かすための試行錯誤をいくつも積み重ねたところで，根本原因を突き止めた上での対応策でなければ，同じようなエラーが生じないことを保証できないからです．

筆者は，原因を突き止めるためには，システマティックなアプローチが必要だと考えます．それをまとめると，

1）まず全体を流れでとらえる

2）客観的な設問を積み上げて問題を切り分ける

となります（**図1**）．

● エレベータ事故の例で思考実験

少し前に，エレベータの誤動作により人が亡くなったというニュースがあったかと思います．このような事故が発生したとき，もしエレベータ会社の社長が「うちの製品には問題はない．アフタ・サービスをしている会社に責任があるのだ」と言ったら，どう思いますか．

もし筆者がその場にいたら，「アフタ・サービスを行っている会社によって事故の発生確率に偏りがあるのですか？」と聞きたくなると思います．それを聞きたい理由は以下の通りです．

事故の原因を突き止めるために，全体を流れでとらえる作業から始めてみましょう．エレベータでも，あるいはほかの製品でも，まず「設計」し，工場で「製造」して，実際に使われ始めてから「保守管理（アフタ・サービス）」を行う，というプロセス自体はだいたい同じであると考えられます（**図2**）．原因の究明というのは，この一連のプロセスのうち，どこに問題があったのかを見極めていく作業にほかなりません．

設　計 → 製　造 → 保守管理 →
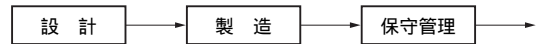
図2　エレベータに関する思考実験

エレベータが運用されるまでにはどのようなプロセスが考えられるだろうか．流れで大まかにとらえると，エレベータをどう作るかを決める「設計」，実際に工場でエレベータを生産する「製造」，そして実際に使われ始めてから「保守管理（アフタ・サービス）」を行う，という三つのプロセスに分かれる．犯人探しの前に，どこが原因で事故になったのかという切り分けが必要である．

次に，問題の切り分けに移ります．もし設計工程に問題があるのであれば，出荷されてくるすべてのエレベータに等しく欠陥が含まれているわけですから，どこで誰が使おうと常に事故が発生する危険性があることになります（事故が発生する確率は全体でそろっているはず）．逆に，保守管理に手抜きがあって事故につながったのだとすれば，保守管理を担当している事業所や会社によって偏りが生じていると考えられます．

つまり，事故の発生確率に特徴的な偏りがないかに着目するということです．もちろん，エレベータ事故のようなものが起きれば，事故の再現調査が最大の決め手になることは言うまでもありません．しかし，その結果を待たないまでも，事故の発生確率に着目することで，ある程度の切り分けが進められるのではないか，ということです．

特にこのエレベータ会社が国際的に営業を展開していて，各国に製品を販売しているとしたら，保守管理を担当する
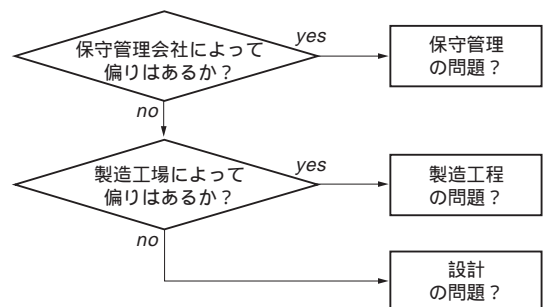
図3　事故原因の切り分けの一例

設計に誤りがあれば一様に事故につながる．逆にそれ以外の製造や保守管理に事故の原因があれば，エレベータの個体によって事故の発生確率が異なるはず．ここに注目して，事故の発生確率に何らかの偏りがないかを調べて当たりをつける．

今回の重大クレームを踏まえた対策骨子

① 顧客満足の向上
（ア）常にお客さまの視点で考える
（イ）顧客苦情は成長の契機ととらえる

② コミュニケーションの改善
（ア）自分たちだけの思い込みは禁物
（イ）ホウ・レン・ソウを大切に
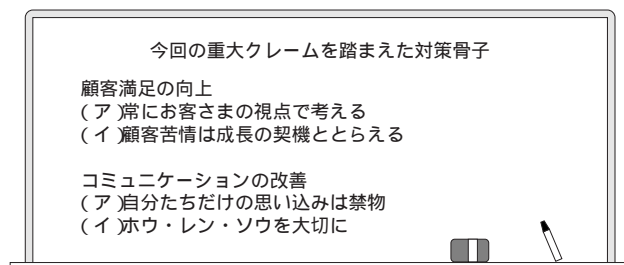
図4　大反省会にありがちな「解決策」

「顧客を重視し，コミュニケーションを円滑化しろ」というのは反論し難い結論である．しかし，問題を切り分けずにこの結論に飛びついた場合，同じ反省の弁をまた述べることになりかねない．

図5　日付でトレースして切り分け

顧客からバグ報告がなされてから最終的に問題が収まるまでどういう経過をたどったのかを時系列で追ってみる．

会社が複数関係しているでしょうから，このアプローチは有効なはずです．もし，調査の結果，ある国の，ある保守管理会社の担当分だけ事故が突出していたのだとしたら，いかに「潔くない」と言われようとも，この保守管理会社を疑わないわけにはいきません．逆に，全く偏りなく世界中で事故が起こっているのであれば，エレベータの設計そのものを疑わないわけにはいきません（**図3**）．

なお，筆者はエレベータ関連の仕事をしたことがないので，これはあくまで組み込みシステム一般として考えた思考実験であることをお断りしておきます．

● 顧客クレーム処理を振り返る

エレベータ事故ほど重大なものでなくても，組み込みソフトウェアの開発をしていれば，顧客からバグ報告が挙がり，それに対処することがあると思います．そんなとき，バグ対応に非常に時間がかかってしまい，重大な顧客クレームに発展してしまったという場面を考えてみましょう．あなたなら，どのような再発防止策をとりますか．

ありがちなのは大反省会を開かされ，偉い人にさんざん絞られた揚げ句，「顧客を重視し，コミュニケーションを円滑化する（**図4**）」という結論を導き出して，やっと解放される，というものです（筆者も似たような経験がないわけではない）．

「顧客を重視し，コミュニケーションを円滑化する」こと自体は間違いではありません．しかし，結論があまりに当たり前すぎるのが気がかりです．本当に問題を切り分けた結果こういう結論を採用したのなら効果はあると思いますが，そうでないとしたら，また同じような反省会を開くはめになりかねません．では，どうしたらよいのでしょうか．

筆者ならまず，顧客からの最初のバグ報告がなされてから最終的に問題が収まるまでを時系列で追って，どのような経過をたどったのかを確認し直します（**図5**）．最近は連絡を電子メールでやりとりすることが多いでしょうから，客観的な「証拠」を集めるのは容易なはずです．こうして得られた情報（**図5**）を見ると，次の2カ所について，不自然に時間がかかっていることが気になります．

1）顧客からの第一報から社内のバグ登録までの期間

2）社内レポート作成から最終報告までの期間

図5を基にさらに調査を進めた結果，次のようなことが分かったとします．

- **1）について**：「顧客から第一報を受け取ったとき，顧客が重大バグと認識していたのを確認しそこねた．加えて社内での再現確認に手間取ったため，当初は顧客のシステムの問題ではないかと疑い，自社製品の問題であるという認識を持つのが遅れた」
- **2）について**：「修正第1版を作成したとき，顧客の窓口となっているサポート・チームが念のため動作を確認したところ，別のエラー・パターンで同種のエラーが検出されることが分かり，最終的な修正に時間がかかってしまった」

ここまで来れば，対策はかなり具体的に立てられそうです．「顧客からみたバグ報告の優先度をなぜ間違ったのか」は，具体的にどのような連絡がなされたのかをたどれば，どこで誤解が生じたのかが見えてくると思います．「バグの修正がなぜ中途半端に終わってしまったのか」は，開発チームの切り分けの手法や，修正後の追加テストを見直す必要がありそうです．

「なにやら当たり前の話だな」，と思われた読者もおられるかもしれません．しかし，実際の「反省会」では，客観的に証拠を集めることなく結論を探してしまうことがよくあります．そうなると，原因は他人にあると考えたがるのが
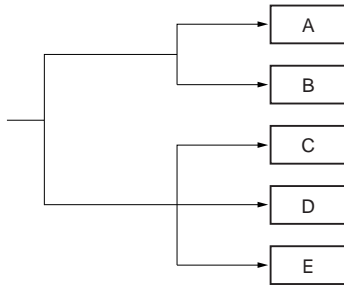
図6　問題を切り分けると樹形図になる

YESかNOかで答えが出る設問を立て，事実に基づいて問題を切り分けていく．これをスマートにfact-based analytical approach（事実に基づいた分析アプローチ）と呼ぶこともある．MECEというのは，この樹形図を作るときのコンセプトである．

図7　MECE分類の落とし穴

MECEの落とし穴をテスト項目の例で示す．②のCブロックに仕様変更があったので，②の内部のみテストし直してリリースしたら，今度は①のBブロックの潜在バグが見付かることもある．MECEというのはあくまで物事を分類する一つのコンセプトであり，万能ではない，ということ．

人間の常ですから，サポート・チームは**2)**を，開発チームは**1)**を主原因と考えるでしょう．これがそのままぶつかれば，不愉快な派閥争いに発展しかねません．結論に飛びつく前にまず事実関係を整理して，客観的な証拠を積み上げるプロセスを踏んで，コンセンサス（共通見解）を得る必要があります．そうしないと，実行可能な対策を得るのは難しいのです．

「ロジカル・シンキング」などと言うと，大げさに聞こえるかもしれませんが，このように，「客観的に切り分けを進めるためのちょっとした工夫」として活用できるものなのです．

● MECEというキー・コンセプト

「客観的な設問を積み上げて問題を切り分ける」というコンセプトを図式化すると，**図6**のようになります．このような樹形図を描きながら問題点を絞り込んでいくのが，以上のすべての議論に共通して言えることです．

このような樹形図の枝葉を分けていくときのポイントは「無駄なく，漏れなく」です．ロジカル・シンキングの世界では，これをMECE（mutually exclusive and collectively exhaustive）と呼ぶのが一般的です．

● ロジカル・シンキングの限界：MECE分類の落とし穴

さて，ロジカル・シンキングに関する一般書であれば，「このように論理的に物を考えることは重要であり，従来の日本人にはこの資質が欠けていた．これからのボーダレス社会においては論理的に考えられるビジネス・パーソンが生き残っていくのであり，ロジカル・シンキングは欠かせない」などと結んで終わりになるような気がします．しかし筆者はあまのじゃくなので，ロジカル・シンキングの限界についても考えてみたいと思います．

組み込みソフトウェア開発の分野で，MECE分類というコンセプトが見かけ上もっともぴったりするのは，なんと言ってもテスト項目の分類です．テスト項目はまず大きく分類し，そこから小さなテスト項目に分けていって，具体的なテストに移ることが多いです(1)．このように整理されたテスト項目の一覧は，MECE分類の分かりやすい一例と言えるでしょう．

さて，テスト項目を分類した結果，**図7**のようなA～Eの機能ブロックに分類できたとします．製品を開発し，テストを終えてリリースした後に仕様変更が入り，Cのブロックを改版したとします．このような場合，Cのブロックに関するテスト項目の再テストのみを行ってほかのテストは省略する，ということがしばしば行われます．

しかし実際には，Bのブロックにもともと潜在バグがあって，それがCブロックの旧仕様ではたまたま救われていた，ということが起こりえます．つまり，テスト・ケースの階層構造のみに着目してテストの絞り込みを行った場合，テスト漏れが生じる恐れがあります[注1]．

実際に組み込みシステムを作ってみると，よく設計されたシステムであっても個々の機能ブロックにはある程度の依存関係があります．ブロック間でやり取りするデータのフォーマットがそろっていなければならないことはもちろんのこと，入出力のタイミングなどもハードウェアが関係するときは特に重要となります．もちろん，それらを注意

注1：「それはテスト・ケースに無駄や漏れがあった，つまりMECEじゃなかったからだ．完ぺきに作られたテスト・ケースならそんなことは起きない」などと言ってMECE分類を擁護する人がいたら，もはやその人は「MECE病」と言えるだろう．

深く設計して製品を作り上げるのですが，仕様変更などが入ると，それが思わぬケースで崩れることがよくあります．設計段階で想定していなかったようなものであれば，テストを考えたときにも見落としているかもしれません．完ぺきに作ったはずの分類でも，分類した細目間に「見えないリンク」[2]があることは避けられません．そこで，大切なのはむしろ，それを見落とさないようにする工夫です．MECEというのは，あくまで物事を分類するための一つのコンセプトにすぎないのです．

● 理論と実務の埋めがたい隔たり

さて，この連載を終えるにあたって，そもそもなぜ「ロジカル・シンキング」などというコンセプトを持ち出したのか，という話をしたいと思います．

筆者がエンジニアとして実務経験を積んで身に付けたことは，ほとんどすべてが実際の経験から学んだものでした．もちろん，人並みにソフトウェア工学の知識に頼ろうとしたこともあります．しかし，ソフトウェア工学の書籍を見るといつも感じたのは，理論と実際の埋めがたい隔たりでした．最初は「理想と現実は違うのだろう」という程度に考えていましたが，だんだんと既存のソフトウェア工学が説く知識体系それ自体を信用しなくなりました．同僚から盗み取ったものか，自分で考えたもの以外，実際の開発に役に立つノウハウや知識にはなり得ない，と思い始めたのです．

何年かの実務経験を経て開発のノウハウもいろいろとため込んだあるとき，自分の身に着けた知識を何らかの形で理論的にまとめられないか，と考えるようになりました．その中であるとき，自分が行っているアプローチが，以前に書籍で読んだ「ロジカル・シンキング」という考え方に非常によく似ているな，と思い始めたのです．ロジカル・シンキングの手法を今まで身に着けたソフトウェア開発のノウハウに当てはめて考えてみると，いままで曖昧模糊（あいまいもこ）としていたものが，すっきりとシンプルに，他人に伝えやすい形でまとまってくることに気が付いたのです（**図8**）．

図8　経験から学んだ考え方は「論理的思考（＝ロジカル・シンキング）」だった

ロジカル・シンキングは，冒頭でも述べたように，経営課題解決のための思考ツールです．ソフトウェア・エンジニアである筆者が培った知識がなぜ，経営戦略のための方法論と結びつくのでしょうか．　これは一見不思議に思えるかもしれませんが，ロジカル・シンキングの生い立ちを考えれば，それほど奇妙なことではありません．

● 「一般工学」としてのロジカル・シンキング

「ロジカル・シンキング」というコンセプトは，経営コンサルティングを行う会社が世の中に広めた考え方です．経営コンサルティングの業界は主として米国で発展し，100年くらいの歴史があります．その中で経験的に培われたノウハウが，ロジカル・シンキングなどのコンセプトに集約されて，まとまっていきました．

経営コンサルタントというとビジネス・スクール出身者ばかりのように思えますが，過去の歴史の中で活躍したコンサルタントを見ると実は，もともとエンジニア出身の人が多くいます[注2]．現在でも，工学系出身の人は意外に多いことが分かります[注3]．「ロジカル・シンキング」の背後には，エンジニアの物の考え方，工学的な考え方が，実はかなり色濃く影響しているのです．

データに基づいて物事を切り分けていき，思うような結果が現実に得られるようにするやり方は，どんな工学分野においても必須と言えます．とは言え，高度な数理科学を駆使した工学の専門知識が経営課題の解決にそのまま使えるわけではありません．揺らん期のコンサルタントたちが応用したのは工学の専門知識ではなく，それらの背後にある物の考え方，つまり，物事を切り分けて解決するアプローチそのものでした．データに基づいて理論的に問題を切り分けるための本質的な考え方，エンジニアに不可欠なこの考え方を分かりやすく定式化していくことで経営課題の解決に応用していったのが，「ロジカル・シンキング」に代

注2：経営コンサルトとして名の知られた大前研一氏の著作には，「MIT（Massachusetts Institute of Technology）で原子力工学を学んだときの経験が，経営コンサルタントになってから役立った」という話が繰り返し出てくる[3]．

注3：例えば，ボストン コンサルティング グループのWebサイト[4]によると，スタッフの23％が工学系の学部出身者であるという．

表される問題解決のツール群なのです．

さてここで，工学という学問の世界に目を移してみると，機械工学や電気工学などの個々の工学分野はありますが，「エンジニアの考え方」を一般的に研究するような「一般工学」のような学問分野は存在しないことに気付きます．もちろん，そのようなものを作ろうとする試みは数多くあるのですが，そのようなことを試みる人たちは皆，何らかの工学分野に精通した上で自分の経験を一般化して述べています．「工学的な考え方そのもの」は，それだけでは研究対象になりにくいのです．

ここで筆者が強調したいのは，「ロジカル・シンキング」は，工学的な物の考え方を経営課題の解決に応用することで，学問の形態をとらないにも関わらず「工学的な物の考え方一般」を定式化しえた，奇妙な成功例だったのではないかということです．

● 工学とは何か：理論と実務の対立を超えるための問い

ソフトウェアの世界においても，理論と実務の対立は存在します．特にソフトウェア工学は，その歴史の短さを考えればまだ揺らん期にあると言わざるを得ません．

理論と実務の対立がある中で，それを乗り越えるために必要なものは何でしょうか．その問いに答えるためには，個々の知識よりも，それに先立つ「工学的な物の考え方とは何か」という根本的な問題を見つめ直すことが何よりも不可欠であると筆者は考えます（**図9**）．この視点が欠落していると，実務家の提供する文章は，単なる使い捨てのノウハウ集に終わってしまうからです．

では，「工学的な物の考え方とは何か」という立場から組み込みソフトウェア開発を眺め直すとどうなるのでしょうか．それに対する筆者なりの回答がこの連載であり，ロジカル・シンキングはそのためのツール（道具）だったのですが，この試みがどの程度成功したのかは，読者の判断を仰ぐほかありません．

連載を続ける中で，「記事が役に立った」というありがたい感想をいただくこともありました．そのような読者の方は，筆者が紹介したノウハウの背後にある「物の考え方」にまで理解を及ぼしていただきたいと思います．実務家である読者の方々は，ソフトウェアの教科書には絶対出てこないような課題にぶつかることがしばしばあると思います．前例のないものであっても，解決すべき問題は多くあります．そのようなときに，この記事が役に立つことがあれば幸いです．

図9　表面にとらわれず，根本的な問題を見つめ直そう

参考・引用*文献

(1) 冴木 元；「ひたすら流すだけ」にさようなら，Design Wave Magazine，2007年1月号，pp.119-124，CQ出版社．
(2) 畑村洋太郎；失敗学のすすめ，p.67，講談社，2000年11月．
(3) 大前研一，斉藤顕一；実践！問題解決法，p.19，小学館，2003年6月．
(4) ボストン コンサルティング グループ；スタッフの出身学部構成（大学），http://www.bcg.co.jp/recruiting/talent.html

さえき・はじめ

<筆者プロフィール>
冴木 元． システム・エンジニア．10回続いたこの連載も今回で終了となります．この回で連載を止めるのは雑誌との「お約束」で，別に，筆者が原稿を落として継続困難になったからでも，ネタ切れを起こしてギブアップしたからでもありません．またお目にかかる機会もあるかと思いますので，その時は再びご愛顧願いたいと思います．今まで読んでいただいた方々，どうもありがとうございました．